

# The Importance of Generalization in Automated Proof

Ken McMillan  
Microsoft Research

Aws Albarghouthi  
University of Toronto

# Generalization

- Many reasoning methods rely on generalization from particular cases for their performance
  - SAT/SMT solvers
  - Abstract interpretation
  - CEGAR, lazy abstraction, etc.
  - Interpolation, IC3, etc.
- In this talk, I will argue:
  - The *evidence* for these generalizations is weak
  - This motivates a *retrospective* approach: revisiting prior generalizations in light of new evidence

# Criteria for generalization

- A generalization is an inference that in some way covers a particular case

Example: a learned clause in a SAT solver

- We require two properties of a generalization:
  - Correctness: it must be true
  - Utility: it must make our proof task easier

A useful inference is one that occurs in a simple proof

Let us consider what evidence we might produce for correctness and utility of a given inference...

# What evidence can we provide?

- Evidence for correctness:
  - Proof (best)
  - Bounded proof (pretty good)
  - True in a few cases (weak)
- Evidence for utility:
  - Useful for one truth assignment
  - Useful for one program path

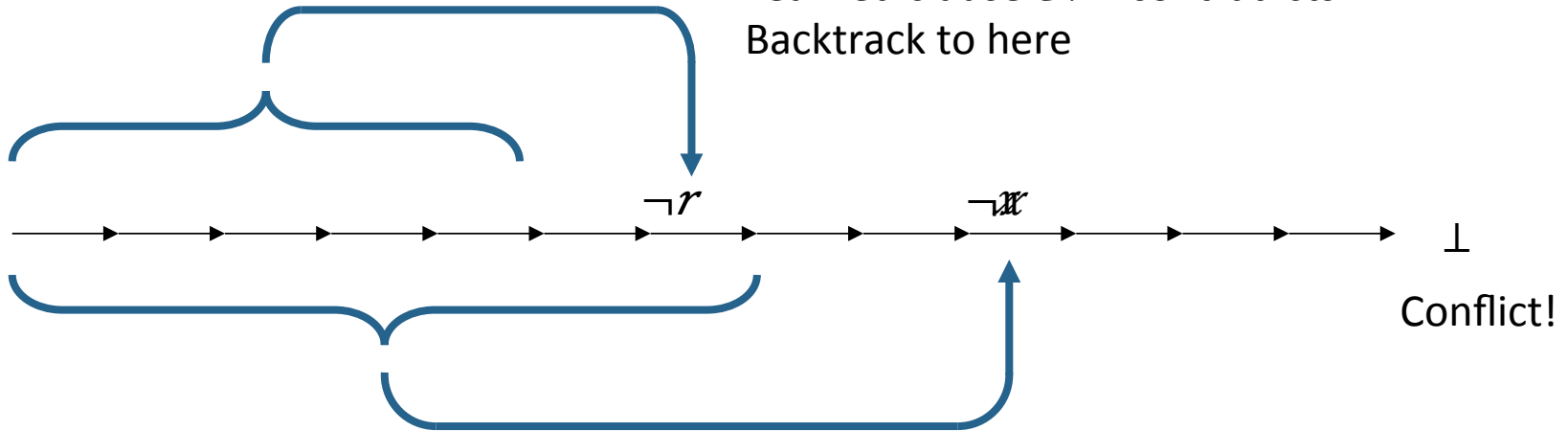
# CDCL SAT solvers

- A learned clause is a generalization
- Evidence for correctness:
  - Proof by resolution (strong!)
- Evidence for utility:
  - Simplifies proof of current assignment (weak!)
  - In fact, CDCL solvers produce many clauses of low utility that are later deleted.
- Retrospection
  - CDCL has a mechanism of revisiting prior generalization in the light of new evidence
  - This is called “non-chronological backtracking”

# Retrospection in CDCL

The decision stack:

Learned clause  $\mathcal{C} \downarrow 2$  contradicts  $r$   
Backtrack to here

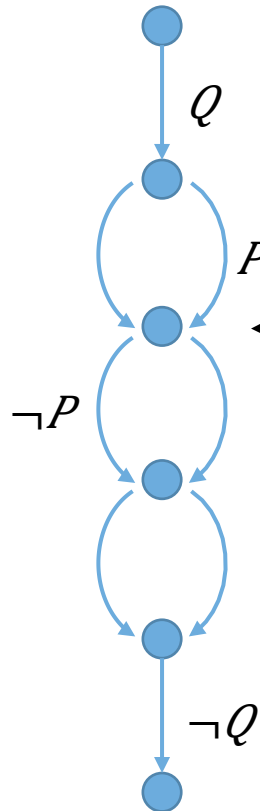


$\mathcal{C} \downarrow 1$  drops out of the proof on backtrack!

Learned clause  $\mathcal{C} \downarrow 1$  contradicts  $\neg x$   
Backtrack to here

- CDCL can replace an old generalization with a new one that covers more cases
  - That is, utility evidence of the new clause is better

# Retrospection in CEGAR



CEGAR considers one counterexample path at a time

Next path can only be selected using  $Q$  of equal utility

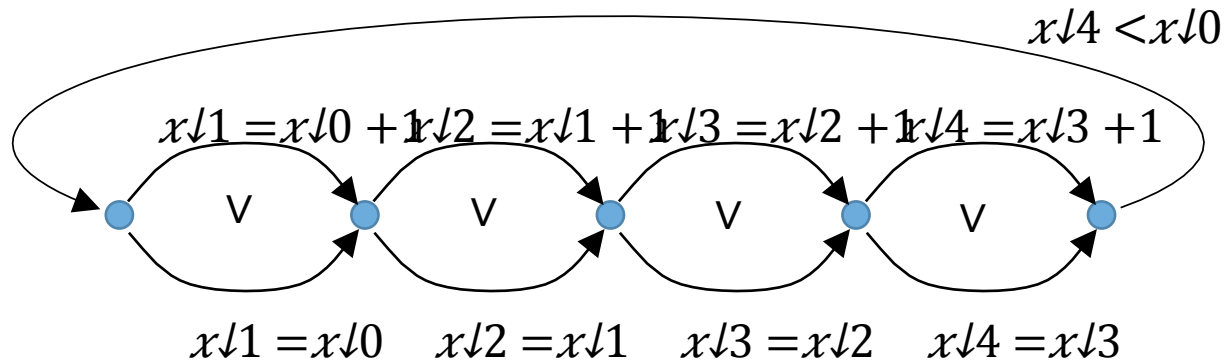
Use predicate  $P$  here

- Greater evidence for  $Q$  than  $P$ 
  - Two cases against one
- However, most CEGAR methods cannot remove the useless predicate  $P$  at this point

# Retrospection and Lazy SMT

- Lazy SMT is a form of CEGAR
  - “program path” → truth assignment (disjunct in DNF)
  - A theory lemma is a generalization
- Evidence for correctness: proof
- Evidence for utility: Handles one disjunct
  - Can lead to many irrelevant theory lemmas
- Difficulties of retrospection in lazy SMT
  - Incrementally revising theory lemmas
  - Architecture may prohibit useful generalizations

# Diamond example with lazy SMT



- Theory lemmas correspond to program paths:

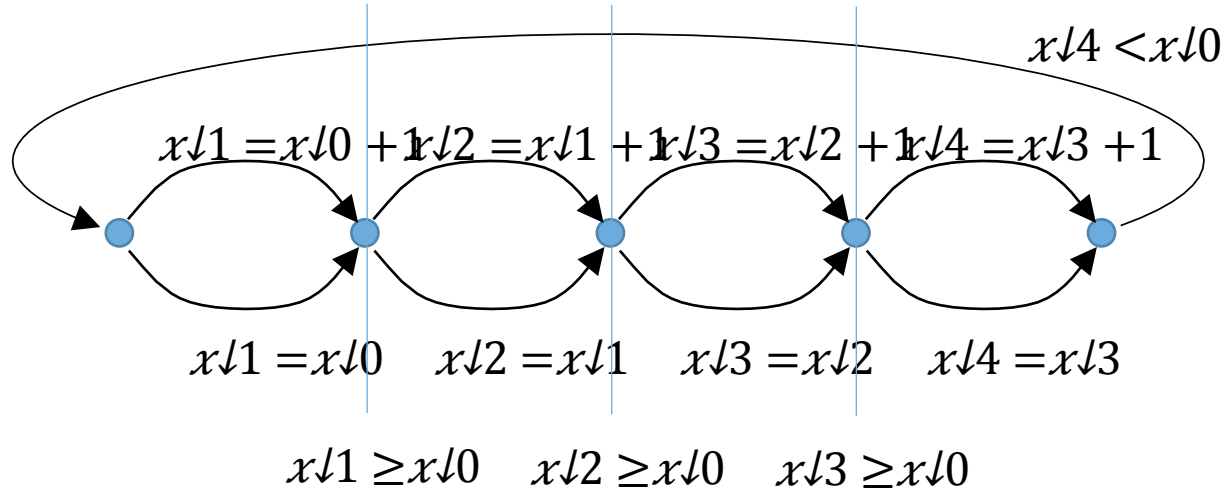
$$\neg(x\l1 = x\l0 + 1 \wedge x\l2 = x\l1 + 1 \wedge x\l3 = x\l2 \wedge x\l4 = x\l3 \wedge x\l4 < x\l0)$$

$$\neg(x\l1 = x\l0 \wedge x\l2 = x\l1 \wedge x\l3 = x\l2 + 1 \wedge x\l4 = x\l3 + 1 \wedge x\l4 < x\l0)$$

... (16 lemmas, exponential in number of diamonds)

- Lemmas have low utility because each covers only once case
- Lazy SMT framework does not allow higher utility inferences

# Diamond example (cont.)



- We can produce higher utility inferences by structurally decomposing the problem
  - Each covers many paths
  - Proof is linear in number of diamonds

# Compositional SMT

- To prove unsatisfiability of  $A \wedge B \dots$ 
  - Infer an **interpolant**  $I$  such that  $A \rightarrow I$  and  $B \rightarrow \neg I$ .
  - The interpolant decomposes the proof structurally
  - Enumerate disjuncts (samples) of  $A, B$  separately.

$S \downarrow A, S \downarrow B \leftarrow \emptyset$

Choose  $I$  so  $S \downarrow A \rightarrow I$  and  $S \downarrow B \rightarrow \neg I$

Chose  $I$  to cover  
the samples as  
simply as possible

Use SMT solver  
As block box

If not  $A \rightarrow I$  then  
add a disjunct to  $S \downarrow A$  and continue.

If not  $B \rightarrow \neg I$  then  
add a disjunct to  $S \downarrow B$  and continue...

$A \wedge B$  is unsatisfiable!

With each new sample, we reconsider the interpolant to maximize utility

# Example in linear rational arithmetic

$$A = (x \leq 1 \wedge y \leq 3) \quad A \downarrow 1$$

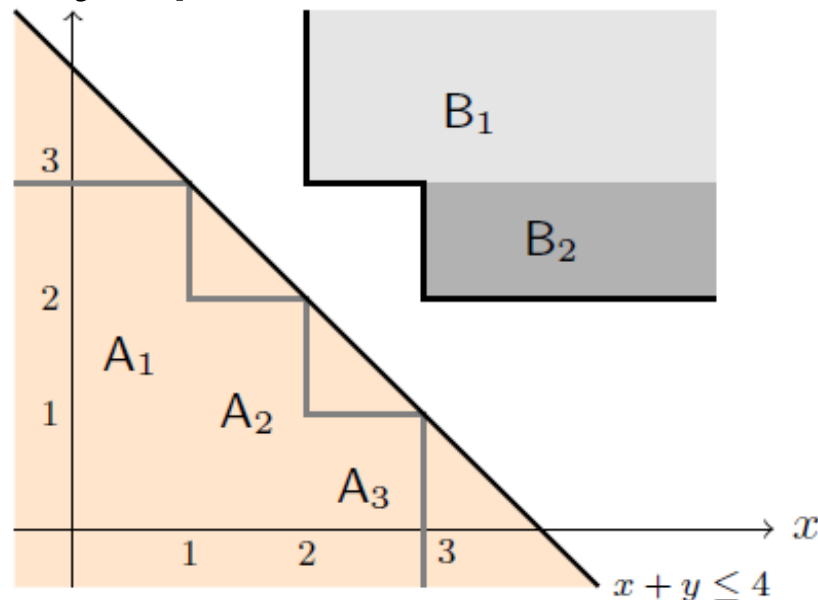
$$\vee (1 \leq x \leq 2 \wedge y \leq 2) \quad A \downarrow 2$$

$$\vee (2 \leq x \leq 3 \wedge y \leq 1) \quad A \downarrow 3$$

$$B = (x \geq 2 \wedge y \geq 3) \quad B \downarrow 1$$

$$\vee (x \geq 3 \wedge 2 \leq y \leq 3) \quad B \downarrow 2$$

- $A$  and  $B$  can be seen as sets of convex polytopes

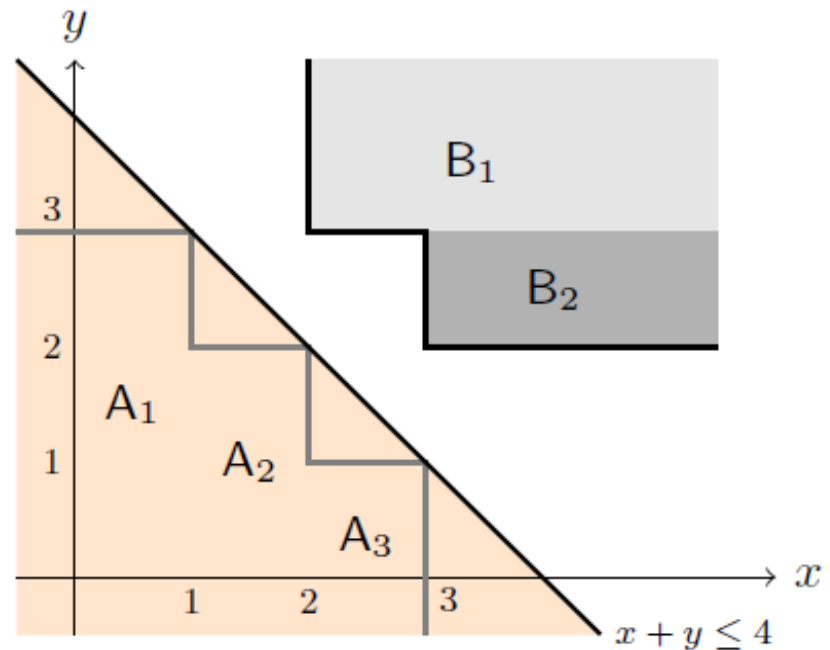
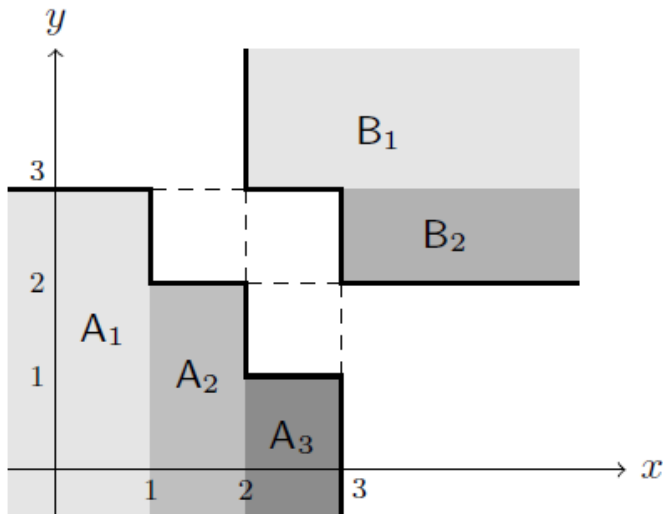


An interpolant  $I$  is a separator for these sets.

# Compositional approach

1. Choose two samples from  $A$  and  $B$  and compute an interpolant  $y \leq 2.5$
2. Add new sample  $A \downarrow 1$  containing point  $(1, 3)$  and update interpolant to  $x + y \leq 4$
3. Interpolant now covers all disjuncts

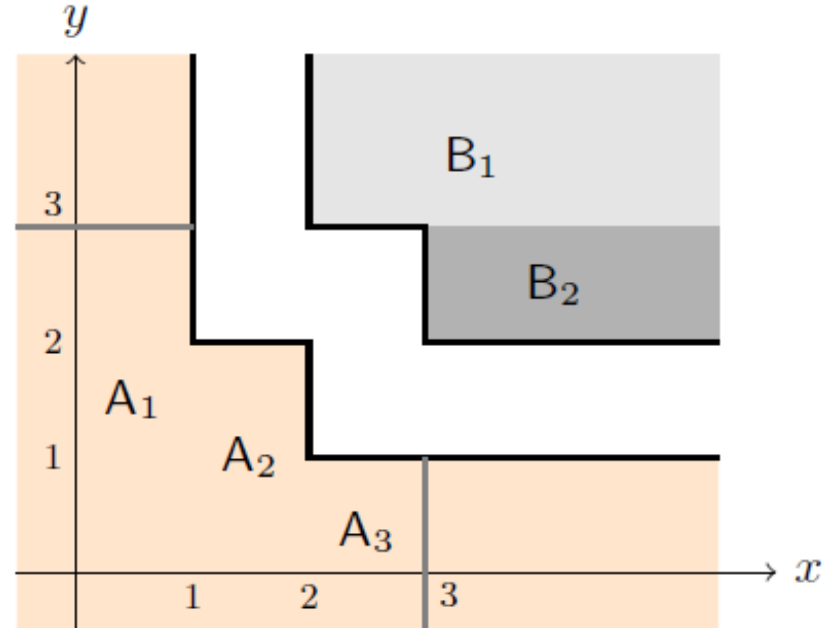
Notice we reconsidered our first interpolant choice in light of further evidence. Point  $(1, 3)$  is in  $A$  but not  $I$



# Comparison to Lazy SMT

- Interpolant from a lazy SMT solver proof:

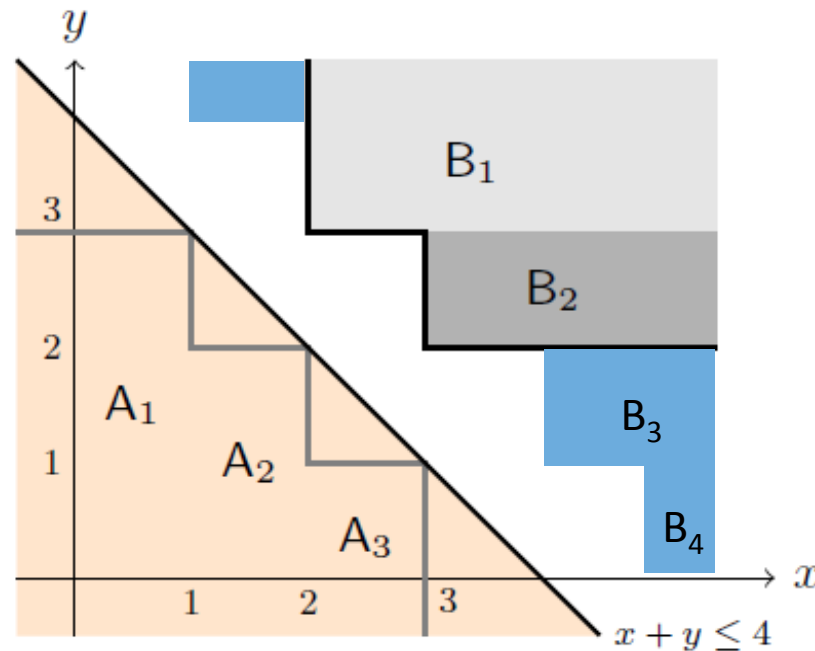
$$\begin{aligned} & (x \leq 2 \wedge y \leq 2) \vee \\ & ( (x \leq 1 \vee y \leq 1) \wedge \\ & ( (x \leq 2 \wedge y \leq 2) \vee \\ & (x \leq 1 \vee y \leq 1))) \end{aligned}$$



- Each half-space corresponds to a theory lemma
- Theory lemmas have low utility
  - Four lemmas cover six cases

# Why is the simpler proof better?

- A simple fact that covers many case may indicate an emerging pattern...



- Greater complexity allows overfitting
- Especially important in invariant generation

# Finding simple interpolants

- We break this problem into two parts
  - Search for large subsets of the samples that can be separated by linear half-spaces.
  - Synthesize an interpolant as a Boolean combination of these separators.

The first part can be accomplished by well-established methods, using an LP solver and Farkas' lemma. The Boolean function synthesis problem is also well studied, though we may wish to use more light-weight methods.

# Farkas' lemma and linear separators

- Farkas' lemma says that inconsistent rational linear constraints can be refuted by summation:

$$\begin{array}{l} a \quad (x - y \leq 0) \\ b \quad (2y - 2x \leq -1) \\ \hline (0 \leq -1) \end{array} \quad \xrightarrow{\text{constraints}} \quad \begin{array}{l} a \geq 0 \\ b \geq 0 \\ a - 2b = 0 \\ 2b - a = 0 \\ -b < 0 \end{array} \quad \xrightarrow{\text{solve}} \quad \begin{array}{l} a = 2 \\ b = 1 \end{array}$$

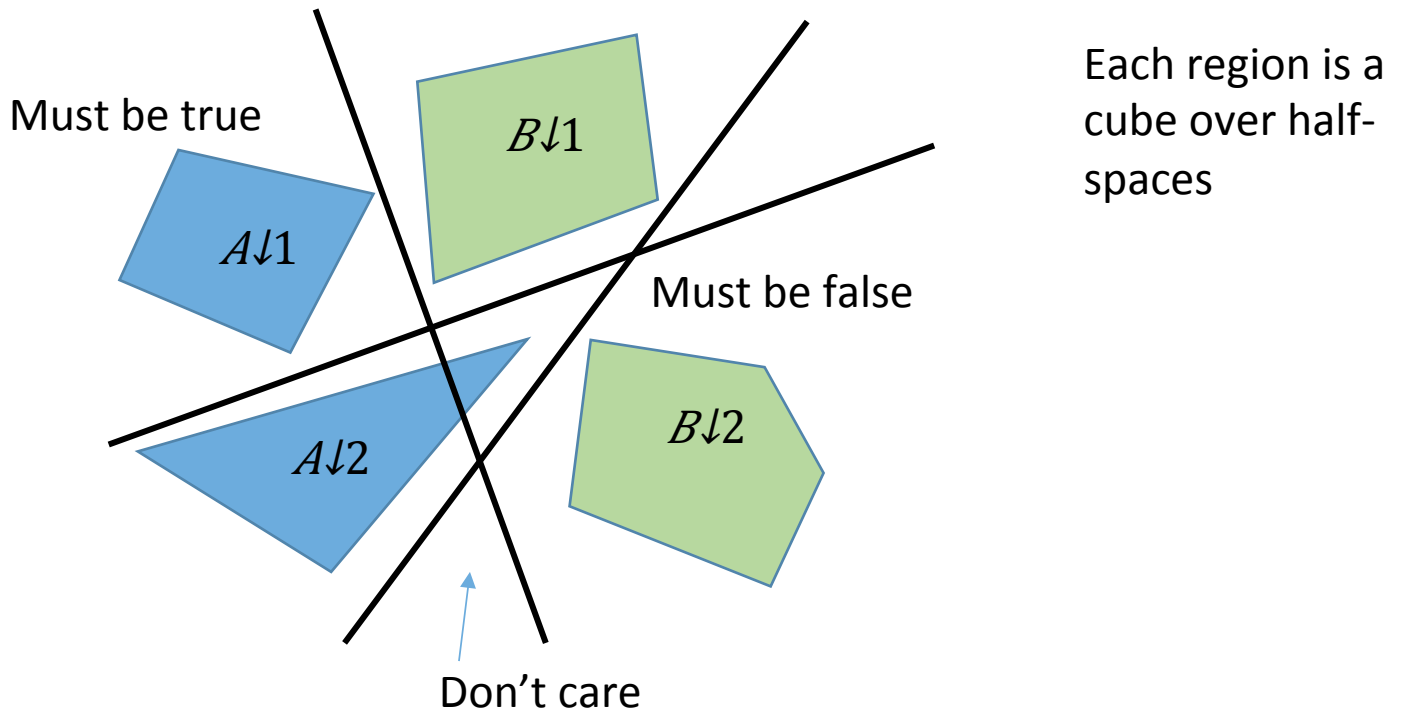
- The proof of unsat can be found by an LP solver
- We can use this to discover a linear interpolant for two sets of convex polytopes  $S \downarrow A$  and  $S \downarrow B$ .

# Finding separating half-spaces

- Use LP to simultaneously solve for:
  - A linear separator of the form  $cx \leq b$
  - A proof that  $A \downarrow i \rightarrow I$  for each  $A \downarrow i$  in  $S \downarrow A$
  - A proof that  $B \downarrow i \rightarrow \neg I$  for each  $B \downarrow i$  in  $S \downarrow B$
- The separator  $I$  is an interpolant for  $S \downarrow A \wedge S \downarrow B$
- The evidence for utility of  $I$  is the size of  $S \downarrow A$  and  $S \downarrow B$ 
  - Thus, we search for large sample sets that can be linearly separated.
  - We can also make  $I$  simpler by setting as many coefficients in  $c$  to zero as possible.

# Half-spaces to interpolants

- When every pair of samples in  $S \downarrow A \times S \downarrow B$  are separated by some half space, we can build an interpolant as a Boolean combination.



In practice, we don't have to synthesize an optimal combination

# Sequential verification

- We can extend our notions of evidence and retrospection to sequential verification
  - A “case” may be some sequence of program steps
  - Consider a simple sequential program:

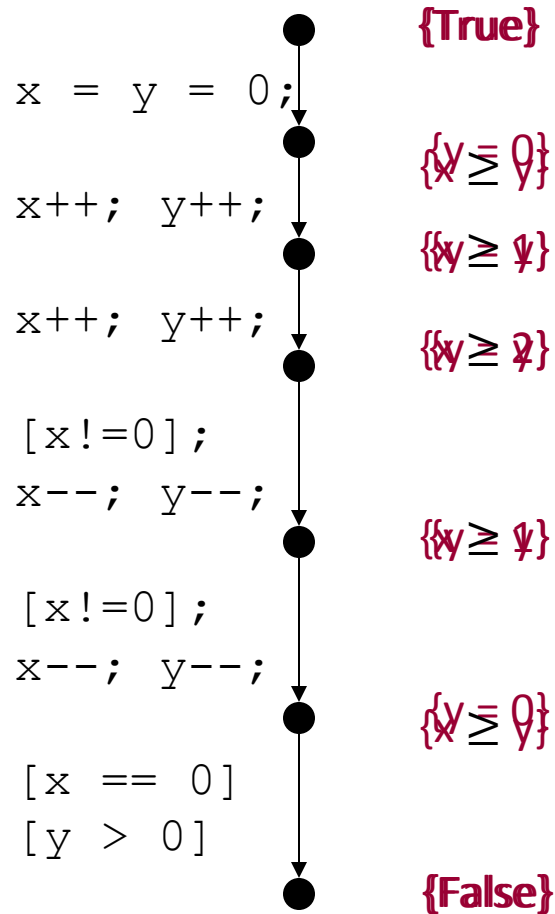
```
x = y = 0;  
while (*)  
    x++; y++;  
while (x != 0)  
    x--; y--;  
assert (y <= 0);
```

Wish to discover invariant:

$\{y \leq x\}$

Two arrows originate from the invariant  $\{y \leq x\}$ . One arrow points to the condition `(*)` in the first `while` loop, and the other points to the condition `(x != 0)` in the second `while` loop.

# Execute the loops twice

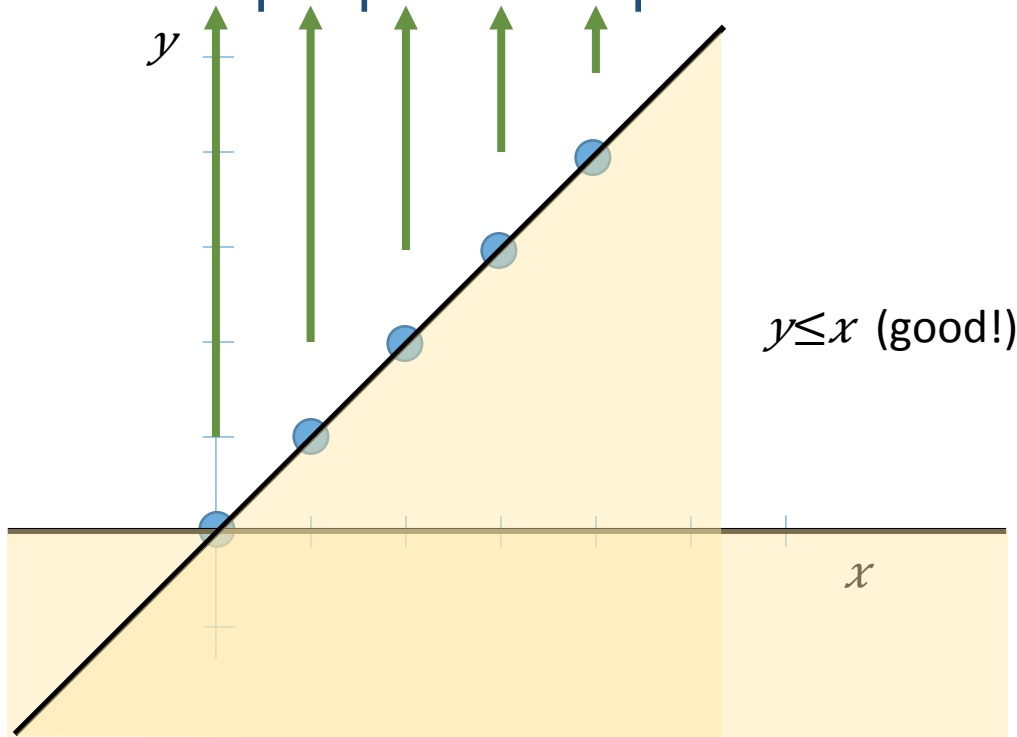


Choose interpolants at each step, in hope of obtaining inductive invariant.

- These interpolants cover all the cases with just one predicate.
- In fact, they are inductive.
- These predicates have low utility, since each covers just one case.
- As a result, we “overfit” and do not discover the emerging pattern.

# Sequential interpolation strategy

- Compute interpolants for all steps simultaneously
  - Collect  $A$  (pre) and  $B$  (post) samples at each step
  - Utility of a half-space measured by how many sample pairs it separates in total.



- Step 0: low evidence
- Step 1: better evidence

# Value of retrospection

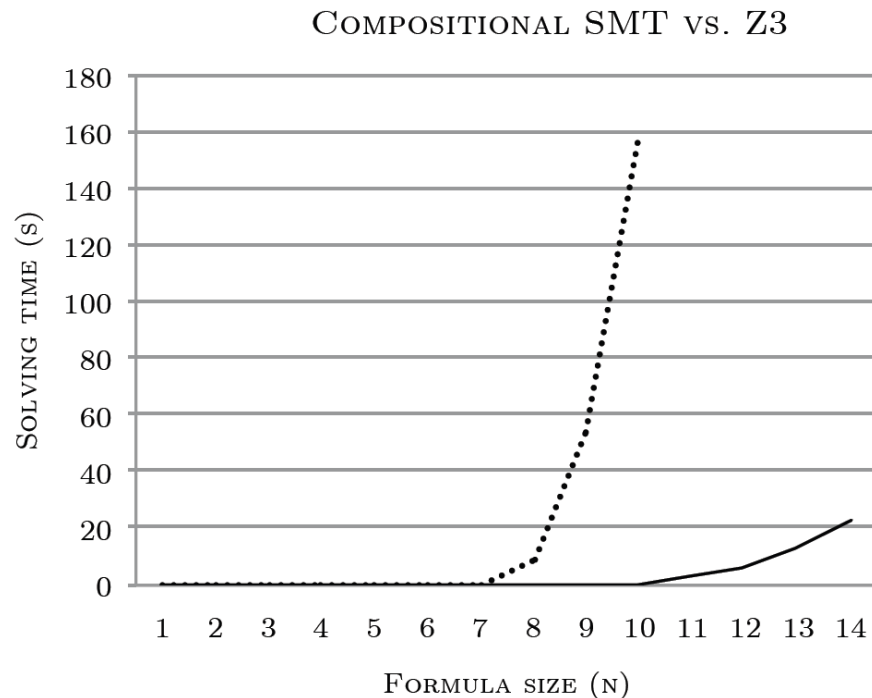
- 30 small programs over integers
  - Tricky inductive invariants involving linear constraints
  - Some disjunctive, most conjunctive

Tool	Comp. SMT	CPA Checker	UFO	InvGen with AI	InvGen w/o AI
% solved	100	57	57	70	60

- Better evidence for generalizations
  - Better fits the observed cases
  - Results in better convergence
- Still must trade off cost v. utility in generalizing
  - Avoid excessive search while maintaining convergence

# Value of retrospection (cont)

- Bounded model checking of inc/dec program
- Apply compositional SMT to the BMC unfolding
  - First half of unfolding is  $A$ , second half is  $B$
- Compare to standard lazy SMT using Z3

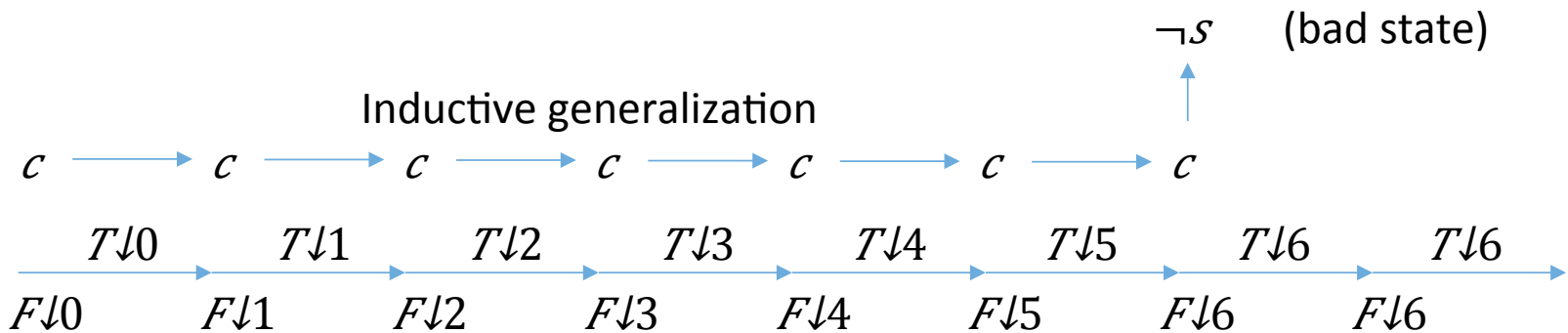


Exponential theory lemmas achieved in practice.

— CSMT  
..... Z3

Computing one interpolant gives power  $\frac{1}{2}$  speedup.

# Example: IC3



- Evidence for correctness: bounded proof
  - New clause may fail to propagate
- Evidence for utility: covers one bad state
- Questions:
  - Is it worthwhile to revisit generalizations?
  - What kind of search procedure can we use?
  - How would the architecture have to change?

For any technique based on generalization, we can ask these questions

# Conclusion

- Many automated reasoning methods rely on generalization from cases
  - Useful to the extent they make the proof simpler
- Evidence of utility in existing methods very weak
  - Usually amounts to utility in one case
  - Can lead to many useless inferences
- Retrospection: revisit inferences on new evidence
  - For example, non-chronological backtracking
  - Allows more global view of the problem
  - Reduces commitment to inferences based on little evidence

# Conclusion (cont)

- Compositional SMT
  - Modular approach to interpolation
  - Find simple proofs covering many cases
  - Constraint-based search method
  - Improves convergence of invariant discovery
    - Exposes emerging pattern in loop unfoldings
- Think about methods in terms of
  - What generalizations?
  - Quality of evidence for correctness and utility
  - Cost v. benefit of the evidence provided

# Cost of retrospection

- Early retrospective approach due to Anubhav Gupta
  - Finite-state localization abstraction method
  - Finds optimal localization covering all abstract cex's.
  - In practice, “quick and dirty” often better than optimal
- Compositional SMT usually slower than direct SMT
- However, if bad generalizations imply divergence, then the cost of retrospection is justified.
  - Need to understand when revisiting generalizations is justified.